SUPLEMENTO MENSUAL DEDICADO A LA IMAGEN SINTÉTICA

Rendermanía

Número 16

Informe Especial
Las extrusiones en Rhino

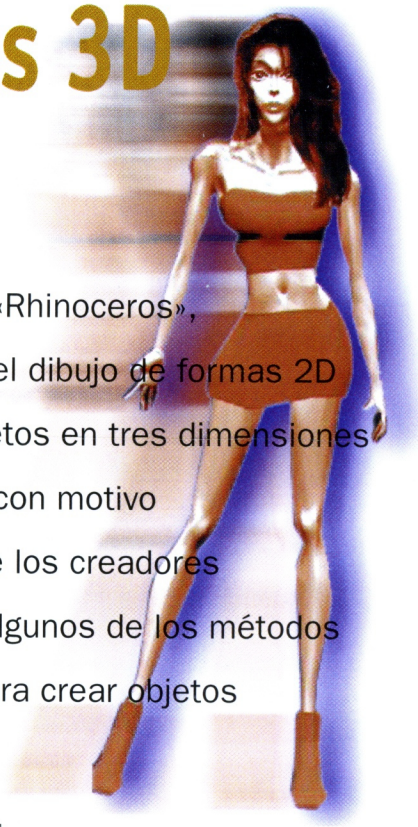
Cómo
Iniciarse en la Povmanía

Foro del Lector
Vuestra galería de arte visual

En el CD
Nueva beta del modelador Rhinoceros

pcmanía

Creación de objetos 3D con Rhinoceros



En el número 11 de Rendermanía presentamos «Rhinoceros», un potente modelador cuya filosofía se basa en el dibujo de formas 2D que luego pueden ser "elevadas" para crear objetos en tres dimensiones (usando una amplia variedad de métodos). Hoy, con motivo de la publicación de una nueva beta por parte de los creadores de«Rhinoceros», aprovecharemos para explicar algunos de los métodos más importantes que emplea este modelador para crear objetos tridimensionales a partir de curvas 2D.

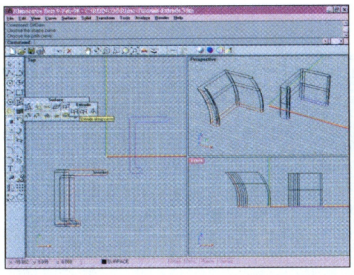
Como ya dijimos entonces, «Rhinoceros» no es un programa de libre uso como POV ni tampoco pertenece a la modalidad shareware. Cuando termine el periodo de prueba de la beta que podéis hallar en nuestro CD-ROM, McNeel y Asociados pondrán el programa a la venta (aunque también cabe la posibilidad de que decidan publicar una nueva beta con otros tres meses de vigencia). Sin embargo, a pesar de esto, hemos creído oportuno publicar la presente beta con el propósito de tener la ocasión de comentar las técnicas empleadas por uno de los mejores mode-

ladores basados en Nurbs del mercado. Aunque ya dedicamos, en el número 11, un informe a «Rhinoceros», dicho informe se ocupó casi enteramente de explicar las opciones de creación y edición de curvas y superficies. Estaba claro que «Rhinoceros» se merecía una segunda parte y eso es lo que encontraréis en estas páginas: un artículo dedicado sobre todo a comentar algunas de las técnicas de "elevación" más empleadas con «Rhinoceros». Esto implica, por supuesto, que deberéis leer el informe que dedicamos a este programa hace cinco meses para poder crear las formas 2D que se emplean como punto de partida en el presente artículo.

Dos extrusiones simples

Cargad el fichero extrude.3dm que encontraréis en el directorio de tutoriales de «Rhinoceros». Una vez hecho esto observad la ventana de perspectiva. En ella veréis dos curvas cerradas que reposan sobre el plano de construcción y una línea roja ligeramente curvada que abandona perpendicularmente dicho plano.

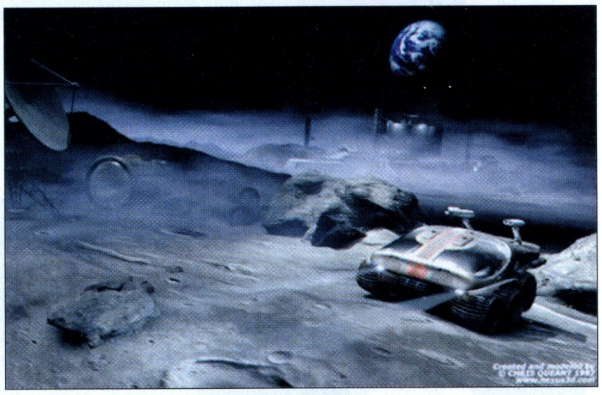
(Nota importante: todas las ventanas de «Rhinoceros» utilizan un plano de construcción sobre el que, por defecto, se colocarán las curvas 2D que iremos dibujando. Cada ventana tiene su propio plano que establece un sistema de coordenadas propio en cada ventana. No es lo mismo indicar un punto con coordenadas 5,5 en la ventana Top que en la Front. Como la orientación espacial de estos planos de construcción puede ser alterada por el usuario, está claro que ello puede ser una gran ayuda a la hora de crear curvas o superficies 2D que tengan una orientación diferente a la establecida por defecto).



Ejemplo de extrusiones sencillas.

Ahora situad el cursor sobre el icono "Surfaces Toolbox" (d5). Si pinchamos sobre este icono con el botón izquierdo del ratón –al que llamaremos de ahora en adelante BI– aparecerá un menú de iconos. Este menú contiene la mayor parte de las opciones de "elevación" de «Rhinoceros». Tened en cuenta que aquí llamamos elevación o extrusión a cualquier proceso por el que se creen formas 3D a partir de curvas o superficies 2D. Pinchad sobre el icono "Extrude Straight" situado en este nuevo menú y luego seleccionad la curva cerrada de color lila. Seguidamente, «Rhinoceros» nos pedirá que indiquemos un punto para establecer el sitio desde el que partirá el patch recto que emplearemos para crear la forma. Marcaremos con BI un punto cualquiera dentro de la curva desde la ventana Top, y después –desde la ventana Right– fijaremos la longitud del patch (lo que haremos desplazando el cursor y pinchando nuevamente con BI). Hecho esto se creará un objeto 3D cuya forma será el resultado de desplazar la curva cerrada inicial a lo largo del patch recto que hemos dibujado. Este es el tipo de extrusión más simple posible.

Si ahora ordenamos un render desde la ventana de perspectiva nos percataremos de que el nuevo objeto carece de tapas. Esto se debe a que hemos extrusionado una curva, no una superficie. Sin embargo, podemos añadir las tapas fácilmente. Para ello eliminad la curva inicial (seleccionándola y pulsando la tecla de supresión). Luego haced aparecer nuevamente el menú emergente de la "Surfaces Toolbox" y pulsad con BI sobre el icono "From planar curves". Esta opción sirve para crear superficies a partir de curvas cerradas (y podremos utilizarla en muchos otros tipos de elevación para poner tapas a los objetos). Ahora, situaos en la ventana Perspective o en la Right y pinchad sobre uno de los extremos no cerrados del objeto. Después repetid toda la operación con el otro extremo. Ya sólo nos queda crear un único objeto a partir de los tres que tenemos por ahora (la forma 3D resultante de la extrusión y dos tapas). Esto

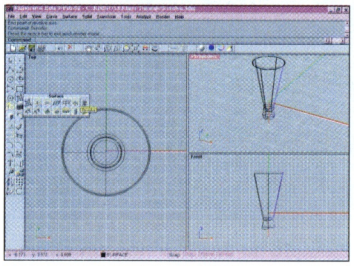


lo haremos pinchando primero sobre el icono "Join" (i8) y luego sobre las dos superficies y el objeto. Así, después de pulsar enter, tendremos un único objeto. Ahora veamos otro caso sencillo de extrusión: dejad pinchado un instante el icono "Extrude Straight" de la "Surfaces Toolbox" para hacer aparecer el pequeño submenú de extrusión. Entonces pulsad sobre "Extrude along Curve" (extrusionar a lo largo de una curva) y después pinchad primero sobre la curva cerrada roja y luego sobre la curva abierta del mismo color que arranca de la anterior. El resultado será un objeto similar al obtenido anteriormente pero que se arqueará siguiendo el patch.

Por alguna razón el objeto obtenido no siempre se crea siguiendo el camino del patch. A veces lo hace en la dirección opuesta a la del patch. Si queremos asegurarnos de que el objeto se crea a lo largo del patch habrá que pinchar en un punto determinado de éste. Así, en este caso, habremos de hacerlo cerca de la curva a extrusionar.

Las superficie de revolución

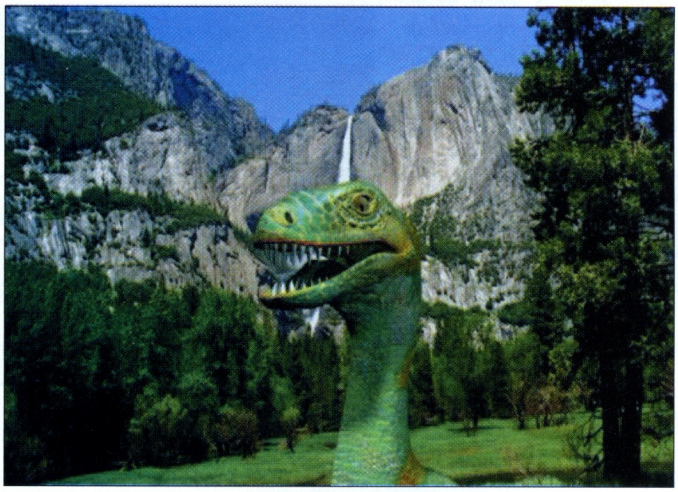
Quienes leyeran el artículo sobre formas orgánicas del número 11 de Rendermanía recordarán que se empleó una superficie de revolución para generar el cilindro básico con el que se creó después la primera cabeza de orco. Como quizá este uso tan poco ortodoxo de estas superficies puede haber



Una superficie de Revolución.

despistado a los lectores más novatos, veremos seguidamente un ejemplo algo más normal:

1) Cargad el fichero de ejemplo revolve.3dm del tutorial.



2) Pulsad sobre el icono "Revolve" del menú emergente de la "Surfaces Toolbox".

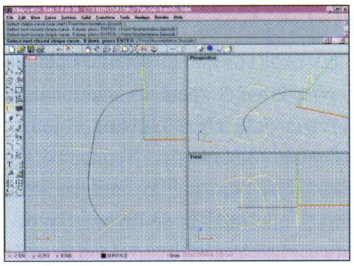
3) Pinchad sobre la curva visible en la ventana Front.

4) Ahora hay que definir el eje en torno al cual girará la forma anterior. Para esto lo mejor suele ser activar antes el grid (icono gridsnap toggle del menú emergente del icono h15). Entonces dibujaremos (en Front) una línea vertical que deberemos dejar pegada a la curva a "revolver". Hecho esto aparecerá un menú en el que se nos preguntará si deseamos que el objeto a crear sea deformable más tarde mediante puntos de control. Si este es el caso marcaremos el apartado "deformable with" y luego indicaremos el número de puntos de control en el recuadro lateral. El ángulo de comienzo para el objeto a crear (start angle) será normalmente de 0 y el ángulo final (end angle) valdrá usualmente 360 grados. Una vez cerrada la ventana se creará, en este ejemplo, un objeto muy parecido a una canasta. El lado inferior del mismo estará cerrado porque hemos colocado el eje pegado a la curva. En caso de haberlo separado un poco, la forma resultante habría tenido un agujero en su base.

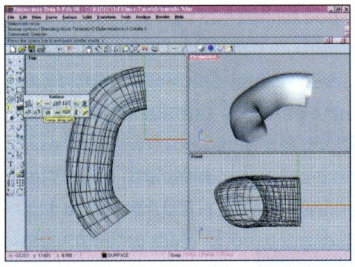
Estas superficies son lo mejor para crear formas como vasos y copas, peones de ajedrez, cañones, floreros, etc. Quienes hayan trabajado con ellas en otros programas no tendrán problemas para manejarlas en «Rhinoceros» y si este no es vuestro caso la experimentación será la única forma de familiarizaros con este tipo de extrusiones.

Barrido a lo largo de un patch

Hasta ahora hemos visto las opciones de extrusión más sencillas de «Rhinoceros». O sea, las que son comunes a la práctica totalidad de los modeladores del panorama infográfico. Pero aho-



Secciones para crear el tubo.

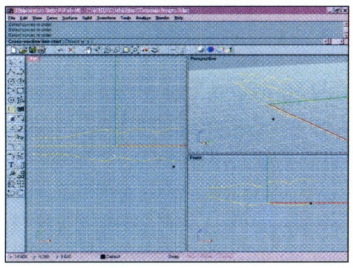


Usando sweep along patch.

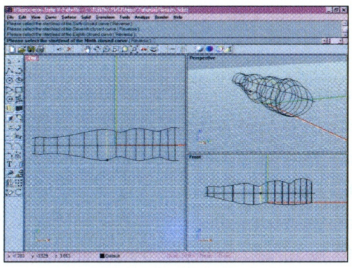
ra comenzaremos a hablar de las opciones que permiten construir formas verdaderamente complejas con este programa.

Comenzaremos por el icono "Sweep along Patch", el cual se encuentra accesible desde el menú emergente de la "Surfaces Toolbox". Para estudiar su funcionamiento realizaremos un par de experimentos. Para el primero utilizaremos nuevamente el ejemplo del fichero extrude.3dm. Una vez cargado este archivo, pulsad en el icono de esta opción y luego pinchad sobre la curva roja cerrada. Cuando lo hayáis hecho el programa os pedirá que pulséis sobre la siguiente curva más cercana a la anterior o que pulséis Enter si no hay más. Esto quiere decir que "Sweep along Patch" puede emplear más de una curva para construir las secciones intermedias de la forma a crear. Luego veremos otro ejemplo donde nos aprovecharemos de esta posibilidad. Por ahora pulsad Enter. Hecho esto «Rhinoceros» nos pedirá que seleccionemos la "rail curve", es decir, el patch. Seleccionad pues la línea roja y pulsad Enter. El resultado será una forma arqueada similar a la conseguida en este mismo ejemplo con "Extrude along Curve", pero con la diferencia de que aquí las secciones intermedias se han orientado siguiendo la dirección del patch. (Aquí llamamos secciones cruzadas a las curvas 2D que emplea «Rhinoceros» para elevar los objetos. En efecto: en todas las opciones de "elevación", «Rhinoceros» crea los objetos gracias a copias o deformaciones –según el caso– de las secciones originales. Estas secciones cruzadas se desplazan en el espacio 3D siguiendo un patch para crear el objeto. Normalmente el usuario dibujará una o más secciones cruzadas para hacer la elevación y el programa generará otras a partir de las del usuario).

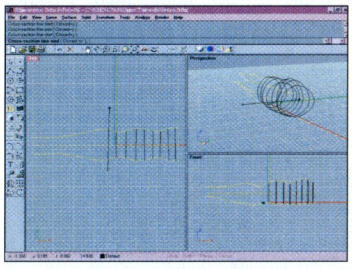
Ahora veamos otro ejemplo. Cargad el archivo de ejemplo barrido.3Dm que hallaréis en el directorio de ejemplos. Observaréis tres curvas cerradas que "Sweep along Patch" utilizará para crear las secciones cruzadas del objeto a elevar. Como veréis, estas curvas son todas diferentes entre sí. Pulsad sobre el icono "Sweep along Patch" y luego pinchad las tres curvas cerradas desde la ventana top siguiendo un orden (por ejemplo, de abajo a arriba). Hecho esto pulsad Enter. Entonces «Rhinoceros» nos pedirá que indiquemos un punto en la primera curva que hemos seleccionado antes. Desde la misma ventana Top situad el cursor cerca del extremo derecho de la primera curva. Cuando lo hagáis aparecerá una pequeña marca en la línea amarilla que representa a la primera curva. Podréis desplazar esta marca por la curva a vuestro gusto pero pulsad con BI cuando la marca esté justo en el extremo derecho de la curva.



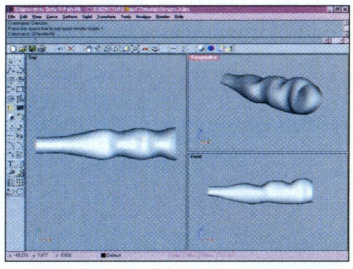
Primero debemos indicar el orden de los perfiles del brazo.



El siguiente paso es poner las marcas a cada sección.



Ejemplo de creación de las secciones del brazo.



Este es el resultado que muestra el brazo acabado.

Entonces haced lo propio con las otras dos curvas. Luego veremos cómo emplea «Rhinoceros» estas marcas. Al marcar la última curva, el programa nos pedirá que seleccionemos el patch a recorrer (la "rail curve"). Pinchad entonces sobre la curva abierta que une a las tres curvas-sección y pulsad Enter. Por último indicad si queréis borrar o no las curvas iniciales y el patch después de crear el objeto.

El resultado –si lo hemos hecho todo bien– deberá ser una especie de tubo arqueado sin tapas en sus extremos (utilizad el render de prueba). Hay que decir que la forma obtenida se extiende siempre desde la forma inicial dada hasta la final. Para comprobar esto deshaced el experimento con undo (ctrl-z) y desplazad a la curva inicial hasta situarla muy cerca de la curva central. Entonces repetid los pasos anteriores para crear la forma 3D. Como veréis, el tubo obtenido no abarca toda la longitud del patch, sino que arranca desde la primera curva y acaba en la última.

Ahora veamos otra cosa. Deshaced el último experimento (con dos operaciones de undo) y volved a repetirlo todo con las curvas cerradas en sus posiciones originales. Lo único que cambiaremos en la nueva elevación será la colocación de la marca en la curva cerrada central. Aquí, en lugar de situar la marca en el extremo derecho de la curva, como en los otros dos casos, la situaremos en el extremo izquierdo. Con esto el resultado que obtendremos al elevar no será un tubo, sino algo muy parecido a un caramelo con su típico envoltorio enrollado. Esto se debe a que «Rhinoceros» utiliza las marcas para saber cómo conectar a las curvas-sección (en esta y en otras opciones de elevación). Normalmente las marcas deberán estar siempre en la misma posición relativa en todas las curvas intermedias o de lo contrario obtendremos resultados indeseados.

El ejemplo del brazo

Es fácil ver que la potente opción explicada en el apartado anterior nos serviría para crear muchos objetos complejos de todo tipo. Teóricamente podríamos, por ejemplo, crear las secciones intermedias de un brazo o de una pierna y usar "Sweep along Patch" para elevar un objeto de este tipo. Es decir, podríamos emplear esta opción si no existiera otra más sencilla para este caso en particular y para otros parecidos. Veamos un ejemplo tomado astutamente del tutorial de Jonathan Block (un usuario de «Rhinoceros»). En este estupendo archivo HTML que podéis hallar en el CD, se explica detalladamente cómo crear un brazo completo, desde el hombro hasta la punta de los dedos. Nosotros, no obstante, nos limitaremos a explicar el procedimiento para crear la forma del brazo desde el hombro a la muñeca.

¿Qué tiene de malo en este caso "Sweep along Patch"? Pues nada salvo que, como podéis imaginar, es mucho más fácil dibujar los perfiles de un brazo que toda una serie de secciones cruzadas que definan la forma del mismo. Por esta razón, «Rhinoceros» nos ofrece un sistema para obtener secciones cruzadas a partir de los perfiles de un objeto. El proceso consistirá pues en obtener primero las secciones cruzadas necesarias a partir de las líneas de perfil y luego en crear al propio objeto a partir de las secciones obtenidas.

Podemos conseguir que «Rhinoceros» genere automáticamente las secciones cruzadas a partir de al menos tres líneas-perfil, aunque las secciones resultarán más precisas si empleamos más líneas. En el brazo de ejemplo creado por Jonathan Black se han empleado sola-

mente cuatro curvas como perfil obteniendo un resultado bastante aceptable. Veamos detalladamente este proceso:

1) Lo primero es dibujar las curvas abiertas que representan a las líneas de perfil del brazo. Dibujaremos dos en la ventana Front para representar el brazo visto de lado y otras dos en la ventana Top para representar el perfil del brazo visto desde arriba. Para dibujar estas líneas podemos emplear cualquiera de las opciones de dibujo de curvas 2D del programa, aunque quizá lo más sencillo sea usar la opción "Sketch Curve" del submenú emergente que aparece con el icono "Interpolated Curve" (d2). El icono "Sketch Curve" nos permitirá dibujar curvas a mano alzada, tal como lo haría un dibujante. Es importante tomar nota de que los perfiles deberán estar alineados, es decir, los dos perfiles laterales de Front deberán estar situados entre los dos de la ventana top y viceversa. En fin, para ahorraros el esfuerzo, cargad el archivo brazo.3Dm que encontraréis en el directorio de ejemplos y observad la forma y posición de las curvas-perfil. Estas curvas no son ninguna maravilla, pero servirán para ilustrar este procedimiento de elevación.

2) Una vez que tengamos preparados los perfiles el siguiente paso será hacer que «Rhinoceros» construya las secciones cruzadas que han de dibujarse entre los cuatro perfiles. Para ello teclearemos el comando Csec. (Nota: «Rhinoceros» admite una extensa lista de órdenes por consola). Hecho esto el programa nos pedirá que seleccionemos las curvas-perfil en orden ("select curves in orden"). Esto quiere decir que habremos de pinchar a las cuatro curvas siguiendo un orden cualquiera, por ejemplo, el del sentido de las manecillas del reloj. Para no confundirnos en este punto lo mejor será hacer esto desde la ventana de perspectiva, haciéndola rotar si es necesario. Una vez que hallamos seleccionado las cuatro curvas pulsaremos Enter.

3) Después, «Rhinoceros» nos pedirá que indiquemos las posiciones donde deberán crearse las secciones cruzadas (con el mensaje "Cross-section line start"). Para ello trabajaremos desde una cualquiera de las dos ventanas de trabajo (la top o la front). Tendremos que ir creando ordenadamente una serie de líneas que cruzarán los cuatro perfiles del brazo. Es importante tomar nota de que estas líneas habrán de ser dibujadas siguiendo un orden, por ejemplo, de izquierda a derecha y de abajo a arriba. También conviene no olvidar que todas las líneas habrán de cruzar los cuatro perfiles. Mencionamos esto porque es posible que al dibujar los mencionados cuatro perfiles no hayamos sido demasiado exactos, de manera que éstos no tengan exactamente la misma longitud. Si alguna de las líneas no engloba a los cuatro perfiles, entonces la sección cruzada correspondiente no se dibujará correctamente. Por cada línea dibujada veremos aparecer una sección cruzada en la ventana de perspectiva. Las secciones correctas deberán quedar enmarcadas dentro de las cuatro curvas-perfil. Cuando hayamos acabado de dibujar las secciones cruzadas pulsaremos return.

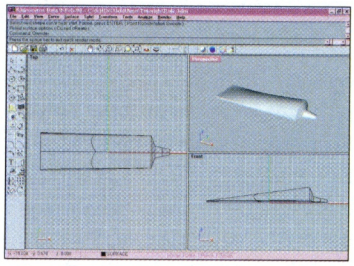
4) Una vez creadas las secciones cruzadas podremos generar el brazo con ellas. Para ello pulsaremos sobre el icono "Lofted" del menú emergente de la "Surfaces Toolbox". Este icono nos permitirá crear extrusiones utilizando más de una sección y funciona de manera muy similar a "Sweep along Patch", aunque "lofted" no precisa de un patch (lo crea el propio programa a partir de las secciones). Entonces «Rhinoceros» nos pedirá que seleccionemos la primera sección cruzada ("select shape curve near start"), lo cual haremos pinchando sobre la primera sección que creamos en el paso anterior. Luego el programa nos pedirá que seleccionemos la siguiente curva cerrada ("select next closed shape curve"). Iremos pues pinchando todas las secciones siguiendo el orden

de su creación hasta llegar a la última, momento en que pulsaremos Enter. Tened cuidado de no seleccionar por error ninguna de las curvas-perfil.

5) Hecho esto, «Rhinoceros» nos pedirá que indiquemos el punto de comienzo/final de la primera sección ("select the start/end of the first closed curve"). Al llevar el cursor hacia esta sección aparecerá una marca que podremos desplazar con el ratón. Esta marca cumple aquí el mismo propósito que ya vimos en la opción "Sweep along Patch". Por ello deberemos colocarla en el mismo punto en todas las secciones (por ejemplo en el extremo inferior de cada sección). Al situar cada marca pulsaremos BI y al concluir con la última pulsaremos Enter, en cuyo momento podremos optar por borrar o no las curvas iniciales. Y con esto ya se habrá creado la forma general del brazo.

El tubo de pegamento

Otro ejemplo que conviene examinar es el descrito en el fichero rule.3Dm del tutorial del programa. En este archivo hay cuatro curvas abiertas que emplearemos para crear un bote de pegamento utilizando la opción para crear superficies "ruled" (estas superficies conectan curvas empleando líneas rectas).



Es un tubo de pegamento creado con ruled.

Pulsad en el icono "rule" de la "Surfaces Toolbox" y luego situaos en la ventana Top. Seguidamente pinchad sobre las cuatro curvas siguiendo un orden dado (por ejemplo de izquierda a derecha y siempre pinchando sobre el extremo inferior de cada curva). Al acabar, pulsad Enter dos veces y decidid si queréis borrar o no las curvas iniciales. Es importante notar que, como en los casos anteriores, será de una importancia vital respetar un orden cualquiera al pinchar sobre las curvas. Si seleccionamos a las curvas salteadamente o si no pinchamos siempre en el mismo punto en cada una, los resultados no serán los esperados.

Operaciones booleanas

Aunque todavía quedan por explicar bastantes opciones de elevación, el espacio se nos acaba y queremos mencionar la posibilidad que nos ofrece «Rhinoceros» de hacer operaciones booleanas. Éstas se hallan accesibles en el menú emergente del icono "Solids Toolbox". Allí, pinchando sobre el icono Subtract, podremos acceder a un pequeño submenú con los iconos Add, Subtract e intersection, los cuales llevan a cabo las operaciones booleanas de unión, resta e intersección respectivamente. Para comprobar su funcionamiento cargad el archivo boolean.3dm del directorio tutorial y realizad algunos experimentos. El sistema para llevar a cabo estas operaciones es bien simple. Primero pulsamos sobre el icono de la operación deseada y luego pinchamos sobre dos objetos. Si la operación en curso es una unión, ambos objetos se convertirán en uno solo. Si se trata de una resta, entonces el volumen espacial del segundo se restará del volumen del primero. Y en caso de que se trate de una intersección, entonces el resultado será un objeto con la forma del volumen compartido de ambos objetos.

Acerca de las ilustraciones

Todas las escenas de estas páginas han sido modeladas con «Rhinoceros» por diversos autores y luego renderizadas desde algún otro programa como «Max», «Lightwave» o «POV».

Empezar con POV (parte I)

Desde hace ya algún tiempo estamos recibiendo cartas de lectores que nos cuentan que POV les parece demasiado difícil como para hacer algo con él. Casi siempre, los lectores que envían estas cartas se encuentran en dos casos bien diferentes: por un lado están los nuevos lectores, recién enganchados al virus infográfico, que suelen hallar incomprensibles los artículos normales que dedicamos a POV en Rendermanía porque no disponen de los conocimientos básicos necesarios (proporcionados por artículos anteriores) y por otro, aquellos rendermaniacos que sí disponen de estos conocimientos pero que encuentran demasiado lento o difícil el modelado con el lenguaje escénico. Teniendo esto en cuenta hemos preparado el presente artículo para demostrar lo fácil de utilizar que es POV y por qué merece la pena emplearlo.

POV es un programa de generación de imágenes sintéticas cuyo motor, basado en el trazado de rayos, permite calcular escenas de una gran calidad. POV es un programa de libre uso y distribución cuyo código fuente se halla también accesible para el público. POV no cuesta un céntimo pero su popularidad no se basa en esto sino en la gran calidad de su render y en las fabulosas posibilidades de su lenguaje escénico. Existen versiones de POV para diferentes sistemas operativos entre los que se hallan MS-DOS y Windows 95

Una obra de Gena Obukhov usando Spatch y Pov.

y todo cuanto vais a leer en el presente artículo es válido para todas las versiones, aunque el autor suele trabajar con la versión 3.02 para MS-DOS.

Funcionamiento general de POV

Para generar las escenas POV se apoya en los llamados ficheros escénicos, unos archivos de texto en formato ASCII que el usuario deberá escribir usando un procesador de textos cualquiera. Estos archivos deberán ser escritos empleando las sentencias y la sintaxis del lenguaje escénico de POV. Una vez que el usuario haya terminado la redacción de uno o más de estos archivos –dependiendo de la estructura del proyecto–, entonces invocará a POV pasándole órdenes específicas acerca de la resolución que se desea para la imagen, el tipo de calidad, la ubicación de las librerías, etc. Estas órdenes, en la versión de MS-DOS, se incluyen dentro de la línea de invocación del programa. Así por ejemplo en la siguiente línea...

povray +icastillo.pov +ocastillo.tga +w320 +h200 -d +v

... el usuario le está pidiendo a POV que procese el fichero escénico castillo.pov y que genere con él una imagen que se guardará en el archivo castillo.tga. La escena tendrá una resolución de 320*256 puntos y durante el proceso de generación la salida a vídeo estará desconectada (-d), aunque el programa nos irá indicando el número de línea en que se está trabajando en cada momento (+v). A menudo estas órdenes son llamadas comandos de la línea de órdenes.

Los archivos escénicos suelen tener la extensión .pov o .inc. Normalmente el archivo principal (el que se da como entrada a POV) suele usar la extensión .pov mientras que los llamados .inc suelen ser ficheros referenciados por el archivo principal o por otros .inc. Esto se hace así por criterios de organización para dividir en varios ficheros a los proyectos demasiado grandes. Así, en el ejemplo anterior, el archivo principal es castillo.pov, el cual podría referenciar a ficheros como torres.inc, almenas.inc, etc., donde vendría la descripción de una serie de objetos que podrían ser empleados en el archivo principal. Todos estos archivos deberán estar compuestos por sentencias válidas del lenguaje escénico de POV, tal y como podría esperarse del fuente de un programa.

Siguiendo con el ejemplo anterior, al invocar a POV, éste cargará el archivo escénico suministrado como entrada y empezará con el proceso llamado de "parsing", durante el cual se comprobará la sintaxis del los archivos escénicos y se construirá a partir de ellos un modelo interno de la escena en un código propio. El "parsing" concluirá cuando este modelo interno –a partir del cual POV va a generar la escena– haya incluido a todos los elementos descritos en los archivos escénicos del proyecto. Este proceso es, pues, muy similar al que realiza cualquier compilador de un lenguaje de programación cuando está procesando los fuentes de un proyecto. Como en este caso, el parser de POV comenzará a leer desde el principio el fichero inicial dado como entrada y continuará procesándolo hasta llegar al final. Si durante el "parsing" POV halla un error de sintaxis, el proceso se detendrá y se mostrará un mensaje indicando el tipo de error y el número de línea donde se halla éste. Entonces, el usuario deberá corregir el error del archivo escénico antes de volver a invocar a POV.

Cuando el "parsing" concluye, POV comienza a generar la imagen pixel a pixel. Esto puede llevar horas e incluso días si la escena es muy compleja o utiliza efectos costosos en tiempo de cálculo, pero no os asustéis: como media, las escenas con los regimientos de orcos del número 14 de Rendermanía requirieron tan sólo de 30 a 45 minutos en un Pentium 150 para una resolución

de 800*600 pixels; lógicamente las pruebas intermedias generadas a una resolución mucho menor requirieron bastante menos tiempo.

Filosofías alternativas con POV

Hay que dejar claro que POV admite varias posibles filosofías de uso. Por un lado es posible crear escenas completas utilizando únicamente POV. Para ello, utilizaremos su lenguaje escénico para crear los objetos y situarlos en el espacio y también para definir las luces, situar la cámara, etc. Pero por otro lado es posible también importar modelos creados desde otros programas como «3D Studio», «Imagine», etc., y utilizarlos para crear nuestras escenas. Esto es factible gracias a que el lenguaje escénico de POV dispone de sentencias como triangle, smooth_triangle y mesh, las cuales pueden ser empleadas por programas de conversión como 3ds2pov, 3dwin y otros. Estas herramientas procesarán el archivo que les suministremos como entrada –conteniendo un modelo creado por un programa como «3D Studio» o «Imagine»– y generarán uno o más archivos .pov o .inc donde se guardará una "traducción" comprensible para POV. En estos archivos la malla poligonal con la descripción del modelo estará descrita con sentencias triangle y smooth_triangle. Estas sentencias se emplean para crear triángulos indicando la colocación de los vértices y no suelen ser usadas "a mano" por los pov-usuarios. (Los miembros del pov-team –los creadores de POV– las han diseñado pensando únicamente en los creadores de tools de conversión).

Aparte de éstas, existen otras formas de emplear a POV. Vamos a intentar resumir las más frecuentes:

1) El usuario emplea el lenguaje escénico de POV para crear la escena completa incluyendo los objetos, fuentes de luz, cámaras, etc.

Una antigua obra del gran Mike Miller.

2) El usuario emplea un modelador creado expresamente para POV y otros programas similares (como Polyray) para construir la escena que finalmente será renderizada desde POV. Estos programas no suelen construir los objetos con polígonos sino con formas matemáticas equivalentes a las primitivas básicas de POV, y su mayor problema es que no suelen representar adecuadamente las operaciones csg (sobre las cuales se sustenta gran parte de la filosofía de construcción de objetos de POV). Programas como Moray y Breeze caen en este apartado.

3) El usuario usa un modelador poligonal comercial como «3D Studio» o «Imagine» para crear los objetos que van a componer la escena. Luego emplea una herramienta de traducción para crear archivos que puedan ser procesados directamente por POV. Algunas herramientas como 3ds2pov traducen no solo la malla geométrica de los objetos sino también la colocación y color de las fuentes de luz, la pigmentación de los objetos, la posición y orientación de la cámara, etc. De esta manera el usuario ni siquiera tendrá que molestarse en trastear con los ficheros escénicos resultantes de la traducción, sino que se limitará a ordenar directamente el render con POV.

4) Aquí, como en el caso anterior, el usuario emplea un modelador comercial y una tool de traducción pero luego edita los ficheros traducidos para situar manualmente a los modelos en la escena y/o crear copias-mesh de éstos (ver número 14 de Rendermanía). También puede optarse por cambiar manualmente a la cámara o a las

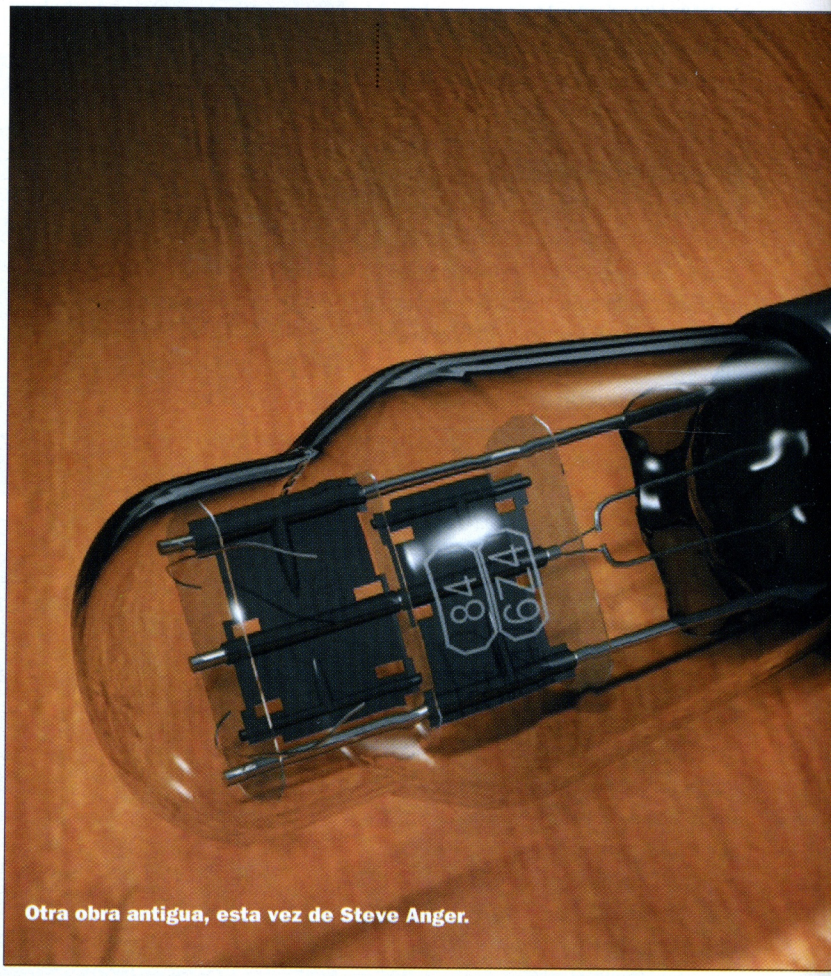
fuentes o por realizar cualquier otro cambio "manual".

5) Últimamente están apareciendo modeladores como «Rhinoceros» o «Spatch» con los que el usuario puede crear objetos empleando splines (lo cual es ideal para, entre otras cosas, crear formas orgánicas). Estos modeladores pueden exportar los modelos directamente al formato de POV (o sea que no se precisa herramienta de traducción). En el caso de «Rhinoceros», éste convierte las formas nurbs de los objetos a mallas geométricas antes de proceder con la exportación mientras que «Spatch» puede exportar los objetos como patches bicúbicos (POV dispone de una sentencia específica para generar estas superficies matemáticas).

Personalmente he probado todas las posibilidades descritas aquí y he constatado que todas tienen sus ventajas e inconvenientes. A menudo suele ser más sencillo utilizar un modelador para crear un modelo complejo como un mech o un orco, pero el lenguaje escénico de POV todavía me sigue pareciendo lo mejor para crear ciertos objetos –sobre todo castillos y edificios diversos–. Últimamente suelo emplear el «Rhinoceros», que se ha convertido en mi modelador favorito, para construir los modelos que posteriormente edito y altero con el lenguaje escénico de POV.

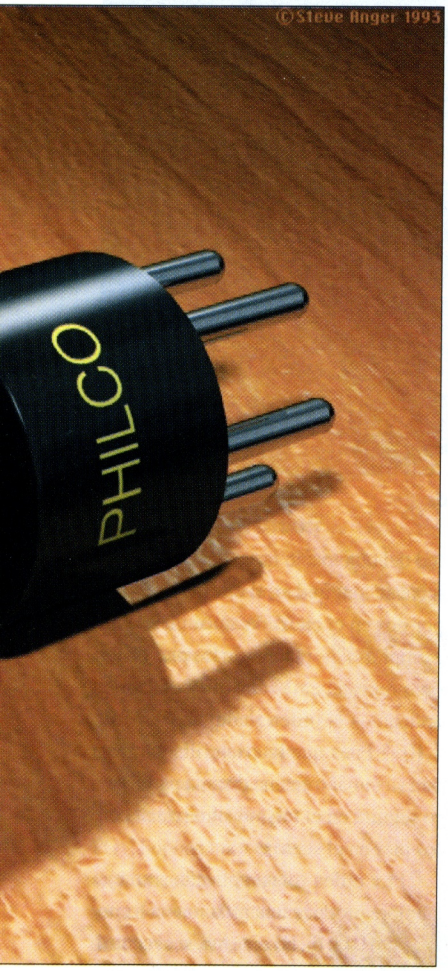
Sin duda los lectores que ya trabajen con programas como «3D Studio», «Imagine» o «Caligary» se preguntarán por qué habrían de molestarse en traducir sus modelos al formato de POV para renderizarlos desde este programa. Pues bien, aquí tenéis varias razones:

1) El motor de render de POV permite crear imágenes de una calidad sencillamente extraordinaria. Con tiempo y habilidad se pueden crear verdaderas obras de arte. Además, la relación calidad/tiempo del programa es también muy buena.



Otra obra antigua, esta vez de Steve Anger.

2) Las sentencias llamadas "de programación" del lenguaje escénico (#while, #if, rnd, etc) pueden procurarnos ventajas fantásticas, aunque probablemente sólo podrán ser apreciadas en su justo valor por quienes tengan buenas nociones de programación. Pensadlo: con programas como «3D Studio», «Imagine» o «Rhino» podemos construir, por ejemplo, un modelo bastante realista de una hormiga con relativa facilidad. Ahora bien, ¿sería igual de sencillo representar un hormiguero? ¿Cuánto tiempo tardaríais en situar cientos de hormigas para crear la escena teniendo en cuenta que deseamos que cada una sea levemente diferente a las demás? ¿Y si también queremos que cada una esté colocada en una postura diferente? Escenas como ésta son –como ya de-



mostramos en el número 14 de Rendermanía– perfectamente posibles con POV. Sin embargo este tipo de escenas, compuestas por cientos de modelos, no son el único ejemplo posible de la utilidad del lenguaje de POV. Es fácil ver que esta potencia en la manipulación de objetos (ya sean creados desde POV o importados) puede ser útil para muchos casos diferentes. Podríamos, por ejemplo, crear un fichero .inc que generase árboles siguiendo un algoritmo determinado para colocar a cientos de objetos simples. Esto ya lo hizo Sonya Roberts con fenomenales resultados. Podríamos también generar un firmamento aleatorio con cientos de estrellas y nebulosas planetarias (Rendermanía número 2) o podríamos colocar los cientos de columnas de un templo usando un buble #while anidado o podríamos...

3) Podemos emplear los pov-ipas para mejorar nuestras escenas. El mejor ejemplo de Pov-ipas es el trees de Sonya Roberts. (Este pov-ipas es un archivo .inc que puede ser referenciado desde nuestros ficheros .inc. El Trees.inc de Sonya crea árboles usando las sentencias de programación de POV). ¡Por otro lado –y a diferencia de lo que sucede con los ipas de «3D Studio»– cualquiera puede crear sus propios ipas para POV! En POV todo está abierto y accesible al usuario. Podemos incluso alterar los fuentes y crear nuestra propia versión de POV si nos viene en gana (aunque el hecho de que esto sea posible no significa que cualquiera pueda lograrlo).

4) El lenguaje escénico de POV puede crear texturas procedurales de una gran calidad.

5) Los "efectos atmosféricos" de POV son potentes y fáciles de emplear.

Seguidamente, vamos a explicar los rudimentos básicos de POV con el fin de que los usuarios de otros programas puedan generar sus escenas con él.

Objetos en el POV-espacio

POV puede representar dos tipos de objetos, los que pueden construirse utilizando las sentencias del lenguaje escénico y los importados. En este artículo vamos a ocuparnos únicamente de los objetos importados pero, para poner un ejemplo sencillo, veamos cómo crearíamos una esfera utilizando el lenguaje de POV:

```
sphere {<0, 1, 2>, 4 pigment{color Red} }
```

Con esta línea le diremos a POV que cree una esfera de color rojo cuyo centro estará situado en las coordenadas <0,1,2> y que tendrá cuatro unidades de radio. La expresión <0,1,2> indica unas coordenadas dentro del universo virtual de POV y se refiere a los ejes X, Y y Z. El punto <0,0,0> es el centro del pov-espacio, el cual se extiende teóricamente hasta el infinito en todas direcciones. Es en este punto donde se cruzan los tres ejes y normalmente el usuario suele disponer a los objetos de la escena cerca de él. Ahora bien, ¿cómo nos orientaremos dentro de este universo? Pues de la manera en que queramos ya que cualquier orientación es arbitraria.

La mayoría de los povmaníacos, sin embargo, utilizan el plano formado por los ejes X y Z como suelo y lo sitúan a la altura de Y=0. Así, por ejemplo, los orcos del número 14 de Rendermanía están orientados perpendicularmente con respecto al plano de "suelo" X-Z y la planta de sus pies reposa a la altura de Y=0. Para visualizar esto imaginad que el centro de coordenadas está situado en el centro de la pantalla de nuestro ordenador.

En este caso el eje X sería una línea horizontal que cruzaría el centro de nuestro monitor y el eje Y sería una línea vertical que dividiría en dos mitades a la pantalla. En cuanto al eje Z podemos visualizarlo como una línea que se introduce perpendicularmente en el plano X-Y del monitor. Las coordenadas de estos ejes tomarán valores positivos o negativos dependiendo del lado

del eje de coordenadas en que se hallen. El lado derecho del eje X es positivo y el izquierdo negativo. El lado superior del eje Y es positivo y el inferior es negativo. Por último, el lado del eje Z que se encuentra "dentro" del monitor es positivo y el opuesto negativo. A menudo en nuestras escenas la cámara se sitúa en algún punto de -Z mientras que los objetos se colocan en el lado positivo de Z. Pero, lo repetimos, esto no es más que una convención.

Dentro de este universo virtual los objetos pueden ser trasladados, rotados y escalados y estas órdenes funcionan igual para todo tipo de objetos. Así, por ejemplo, la línea...

sphere {<0, 1, 2>, 4 pigment{color Red} translate<4,0,10>}

...significa que la esfera originalmente creada con su centro en <0, 1, 2>, será representada en la posición final <4,0,12> al renderizar la escena. Esto sucede así porque la sentencia translate suma un desplazamiento relativo al punto anterior donde estuviera el objeto. Para que esto funcione la sentencia debe quedar dentro de los paréntesis que engloban a la definición del objeto. Podemos utilizar todas las sentencias translate que deseemos para un mismo objeto y de hecho podemos emplear cualquier secuencia de transformaciones. Así, por ejemplo, es válida la línea...

sphere {<0, 1, 2>, 4 pigment{color Red} translate<4,0,10> rotate<45,0,0> scale<2,2,2>}

...donde se realiza primero una traslación, luego una rotación de 45 grados en el eje X y, finalmente, una escalación del objeto por 2 en todos los ejes. (Las operaciones se realizan, claro está, en el orden en que están escritas).

Una obra abstracta de Russel Garwood.

La posición espacial donde se realizan las operaciones de rotación o escalado influye en los resultados finales. Veamos un par de ejemplos:

1) Si un objeto está situado en el pov-espacio de manera que su centro de gravedad se halle en <0,0,0>, entonces, al aplicarle la orden scale<2,2,2>, el objeto duplicará su tamaño sin moverse del sitio. En cambio, si está situado en otra posición antes de la escalación, el resultado –aparte de la duplicación de tamaño– será el de trasladarse de posición. Para comprender esto tened en cuenta que la escalación es una multiplicación de todos los puntos espaciales del objeto. Si el centro del objeto estaba en <0,0,0>, el objeto seguirá estando en el mismo sitio después de la multiplicación pero si estaba, por ejemplo, en <2,3,8>, su nuevo centro pasará a ser <4,6,16> (después del scale <2,2,2>). Naturalmente, las operaciones de escalación no tienen por qué ser idénticas en todos los ejes. Así, por ejemplo, una orden como <2,1,.50> multiplica por dos la longitud del objeto a lo largo del eje X, deja igual su longitud en el eje Y y lo divide por dos en el eje Z. Veamos un último ejemplo: si las plantas de los pies de un orco reposan en Y=0, ¿qué ocurrirá si aplicamos la orden scale <1.10, 1.10, 1.10>? Pues que la criaturilla aumentará su tamaño en una décima parte, pero seguirá en el mismo sitio (o sea con los pies en el suelo).

2) La orden rotate aplicada a un eje siempre produce una rotación en torno al punto 0 de dicho eje. Así, si el centro del objeto está situado en <0,0,0>, entonces cualquier orden de rotación hará que el objeto gire sobre si mismo. Si en cambio el objeto está situado en cualquier otro punto, entonces girará como un satélite en torno al punto 0 del eje de rotación. En cuanto al sentido de las rotaciones, este dependerá del signo aplicado a los grados de rotación. Para saber siempre hacia dónde rotará un objeto hay que aplicar la regla de la mano izquierda: haced el signo de ok con la mano izquierda y orientadla de modo que el pulgar apunte en la direc-

ción positiva del eje en el que queréis efectuar la rotación. Pues bien, los dedos estarán curvados en la dirección positiva del giro. Veamos un ejemplo: tenemos una esfera situada en la posición <8,10,0> a la que aplicamos una rotación de +90 grados en el eje Y. ¿Cuál será la posición final de la esfera? Pues la <0,10,-8>. Para visualizar esto imaginad que cuando se rota en un eje, el objeto girará en el plano definido por los dos ejes restantes.

Ahora pasemos al siguiente punto: cómo manejar objetos importados desde POV.

Objetos importados

Como antes hemos dicho –y salvo en el caso de «Rhinoceros»– las mallas de triángulos de un objeto creado desde un modelador externo deberán ser procesadas con alguna utilidad para generar archivos que puedan ser directamente leídos por POV. Supongamos el caso de un archivo llamado vaca.3ds con el modelo de una vaca almacenado en el formato de «3D Studio». Los pasos para renderizar a esta vaca desde POV serán:

1) Primero procesaremos el 3ds para crear unos archivos directamente digeribles por POV. Para ello la mejor opción sigue siendo el 3ds2pov de Steve Anger (que podéis hallar en el CD-ROM), aunque también podéis emplear otras utilidades como el 3dwin. Con 3ds2pov (que es una utilidad para MS-DOS) la cosa será tan sencilla como teclear "3dspov vaca.3ds vaca.pov". Esto creará dos archivos llamados vaca.pov y vaca.inc. El primero guardará las sentencias que definen a la cámara, las luces y las texturas e invocará al segundo fichero donde se guarda la malla geométrica de la vaca. Como 3ds2pov ha traducido también la cámara y las luces que había definidos en vaca.3ds, esto quiere decir que vaca.pov será directamente ejecutable por POV, con lo cual podremos pasar al siguiente paso.

El fichero 3ds exportado a POV.

2) Invocaremos a POV con la línea "povray +ivaca.pov +ovaca.tga -d +x +v +p +w640 +h480 +a +b192". Esta orden ordenará a POV que genere el archivo vaca.tga con una resolución de 640*480.

Otra magnifica escena de Gena Obukhov

¡Y esto es todo! Sin embargo, si verdaderamente queremos sacar partido a POV, habremos de hacer muchas más cosas. Podríamos añadir un suelo, crear copias del objeto, trastear con las luces, añadir un cielo, efectos de niebla, cambiar la textura de la vaca, etc. Y para ello tendremos que seguir aprendiendo cosas sobre POV. Para ello lo mejor será echar un vistazo a los archivos que hemos generado con 3ds2pov. Si editáis vaca.pov y echáis una ojeada al final del fichero observaréis la siguiente sentencia:

#include "vaca.inc"

La sentencia #include funciona exactamente igual que la directiva del mismo nombre del lenguaje C. Esto es, la sentencia indica a POV que debe leer y procesar el fichero incluido entre comillas. Como este archivo contiene la malla con la vaca, el resultado será que POV representará a la vaca en la posición espacial en que fue definida (no hay translates en el archivo .inc).

La sentencia #include tiene una gran importancia ya que POV volverá a leer y a procesar el archivo "incluido" cada vez que encuentre una orden #include durante el proceso de parsing. Para comprender lo que esto significa hay que tener en cuenta la existencia de otras características importantes de

Una obra de Ian Mackay.

POV como la existencia de la sentencia #declare y la posibilidad de usar variables. Examinad las dos líneas siguientes:

```
#declare posX=10
#declare posX=PosX+10
#declare gY=.20
sphere {<posX*2, 0, 0>, 4 pigment{color Red} scale<1,gY,1>}
```

Después de esto POV creará una esfera roja en las coordenadas <40,0,0>. Así, como podéis ver, #declare puede ser utilizada para crear variables que pueden ser empleadas en sentencias posteriores de creación y modificación de objetos. Lógicamente estas variables seguirán existiendo y conservando su valor aunque se cargue y procese un nuevo fichero usando #include. Como el fichero "incluido" puede hacer referencia a variables definidas en el archivo de llamada, esto, en la práctica, significa que podemos usar las sentencias #include y #declare para crear archivos que funcionarán como si fuesen funciones de C (aunque POV realmente no dispone de sentencias para crear funciones). Esto implica que un mismo fichero podrá generar objetos diferentes en función de los valores que lleguen a sus variables. Este es el fundamento básico en que se apoyan todos los Pov-ipas (como el trees.inc de Sonya Roberts). Finalmente hay que decir que POV permite que los ficheros "incluidos" tengan a su vez sus propias sentencias #include, con lo cual podemos encontrarnos con un caso como el siguiente: POV está trabajando sobre el archivo principal (o sea el .pov inicial) y encuentra un #include al fichero vaca.inc. Entonces carga este fichero y empieza a procesarlo hasta que halla dentro del mismo otra orden #include que referencia al fichero pasto.inc (el cual crea por ejemplo un cuadrado de hierba a los pies de la vaca). Lo que sucederá entonces será que POV terminará de procesar al archivo invocado y al concluir regresará al fichero que ha llamado a pasto.inc. Entonces se continuará procesando a vaca.inc y, al acabar con la última línea de este archivo, se regresará al fichero que invocó a éste aquel (o sea a vaca.pov) y se seguirá trabajando con él hasta llegar a su última línea, en cuyo momento el parsing habrá concluido y comenzará la generación de la imagen.

Pero, siguiendo con nuestro ejemplo, podemos crear más vacas con sentencias como...

```
object{#include "vaca.inc" translate<344,0,400>}
object{#include "vaca.inc" translate<548,0,600>}.
```

La sentencia object se emplea para poder englobar a objetos o a grupos de estos a fin de poder aplicarles transformaciones espaciales o asignarles una textura. Existen muchas maneras de combinar el uso de object con union, #declare y mesh. Pero por hoy se nos acaba el espacio. El próximo mes terminaremos de repasar los aspectos más básicos de POV (cámaras, luces, etc.) para beneficio de los aspirantes a povmaníacos. Después de esto abordaremos otros temas como la creación de texturas procedurales y la animación, los cuales han sido muy poco tratados en Rendermanía.

Las vasijas de Ian Mackay.

Nota sobre las imágenes

Casi todas las imágenes utilizadas para ilustrar el presente artículo han sido creadas por conocidos artistas que emplean POV para renderizar sus escenas.

El Foro del Lector

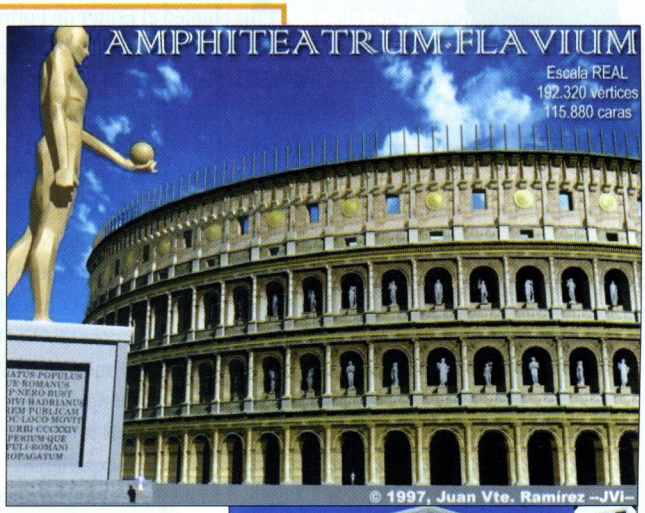
Nota importante. Podéis remitirnos vuestros trabajos o consultas, bien por carta a la dirección que figura en el sumario de Pcmanía, o vía e-mail a rendermania.pcmania@hobbypress.es

Arte Virtual

La lista de temas que hoy ofrece nuestro foro virtual es tan extensa como siempre.Dragones, chicas sintéticas, reconstrucciones virtuales de edificios de la antigua Roma,copias infográficas de cuadros del renacimiento, ejércitos-mesh, habitaciones, iglesias, invasiones extraterrestres, máquinas de guerra para la contienda orco-humana e incluso un magnífico orco cuya apariencia es tan simpática como brutal.

Juan Vicente Ramírez Roda es, aparte de un amante de la informática y la infografía, un gran aficionado a la historia de la antigua Roma y en particular a los edificios y monumentos de aquella época. Por esta razón Juan se ha embarcado en un fantástico proyecto: la recreación virtual de los edificios del Foro Romano con «Caligary TrueSpace». En su carta Juan nos narra cómo intenta reproducir la escala y los detalles de estos edificios lo más fielmente posible. Los detalles son muy interesantes, leed la carta.

Aunque Juan solicita una crítica no me siento capacitado para hacerla. Tus edificios virtuales me han parecido magníficos (sobre todo el coliseo) pero tendría que compararlos con los reales, de los cuales no tengo un conocimiento demasiado exacto (aunque también me gusta mucho la historia, sobre todo la del Imperio Romano Oriental y la de todo el periodo medieval y del Renacimiento). Por ello me limitaré a hacer la valoración que has pedido. Tu proyecto es muy ambicioso pero tus imágenes demuestran que estás plenamente capacitado para llevarlo a cabo. La idea me parece muy buena y creo que todos esperarán con tanta ansia como yo el momento en que puedas enviar escenas con el panorama del Foro completo. ¡Suerte y paciencia!

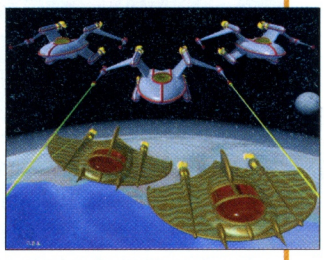


El Foro del Lector

El orco enviado por **José Luis Robledo Pescador** es sencillamente magnífico y su aspecto refleja a las mil maravillas la típica apariencia orcoide. Un estupendo ejemplar de orco que José Luis modeló con «Imagine 4.0» basándose en una miniatura Citadel. Para ello, José Luis empleó el Editor de Formas creando así objetos que posteriormente fueron retocados desde el Editor de Detalles. Sin embargo quizá lo mas destacable sea que la cabeza fue creada partiendo de primitivas simples como esferas, conos y cilindros (¡!). Unos objetos sobre los que José Luis trabajó empleando operaciones Csg y edición de vértices.

José Luis me pide una crítica pero en lugar de ello le daré un pequeño consejo: has sacado un gran partido de técnicas que realmente no son muy adecuadas para crear formas antropomórficas. Creo que para este tipo de modelos es más fácil emplear un modelador basado en splines como «Rhino» o «Spatch»; conseguirías formas de apariencia más suave y con menos problemas en los puntos de unión de las articulaciones. Luego podrías exportar los objetos del modelo a «Imagine» para, desde allí, darles el acabado necesario así como preparar la postura del orco y generar el render. En cuanto a tus preguntas, es posible que dentro de poco volvamos a tratar algunos aspectos interesantes de «Imagine», como los bones y la animación. No hay, que yo sepa, ningún libro sobre «Imagine 4.0» a este lado del charco. Tan sólo tenemos el de Philip Shaddock dedicado a la versión 2.0 editado por Anaya. Por otro lado «Imagine» no tiene ninguna herramienta para reducir polígonos. Para ello tendrías que acudir a alguna utilidad externa; intentaré hallar alguna.

Guillermo Bartle Agustín nos envía escenas de batallas entre cazas espaciales usando como fondo el tema de la invasiones extraterrestres y pide una crítica constructiva. Pues bien, tus escenas, preparadas y renderizadas con POV, son resultonas pero creo que habrías podido mejorar mucho el fondo estelar. ¿Viste el número 3 de Rendermanía? Los fondos que allí aparecían tenían defectos pero mostraban más color y las estrellas tenían diferentes tamaños. También podrías haber alargado los halos que representan el fuego de los cohetes e incluso haberles dado varias capas de color (de rojo a amarillo). Por último, quizá habría sido mejor poner un buen bitmap para la textura de las naves (el típico bitmap con cuadros grises desiguales suele dar buenos resultados). Gracias por tu estupenda catapulta para mis lanceros aunque, como ya habrás visto, ahora tengo un pie puesto en ambos bandos...



No he podido comprobar el fallo que mencionas del ataque planetario 1 porque no has incluido el archivo texturas.inc. Lo sustituí por textures.inc, pero entonces aparecían más errores en el parsing.

Nadie nos había enviado nunca antes una escena intentando reproducir un cuadro clásico y por ello **Armando Gutiérrez Pernía** merece nuestra más sincera felicitación. La escena, creada con «3D Studio 3.0» y «Caligary TrueSpace 2.0» tiene un gran parecido con el cuadro original (del que este autor nos envía una foto); las proporciones de los modelos, las posturas, incluso el fondo, todo es perfecto. Prácticamente lo único que ha faltado ha sido "envejecer" la imagen para hacerla aun más idéntica al cuadro original. Enhorabuena.

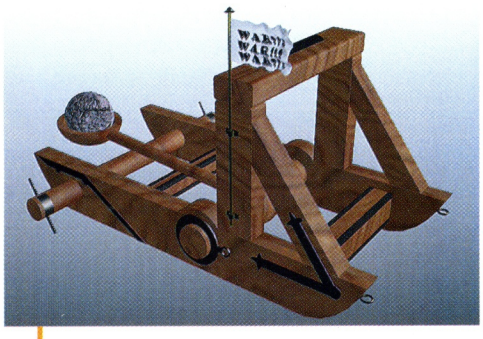
José Francisco Calvo Moreno nos envía una catapulta para apoyar a los orcos. ¡Gracias! Pero ¿por qué no le has puesto ruedas? ¿Cómo demonios van a arrastrarla los orcos al campo de batalla?

En cuanto a tus preguntas e ideas...

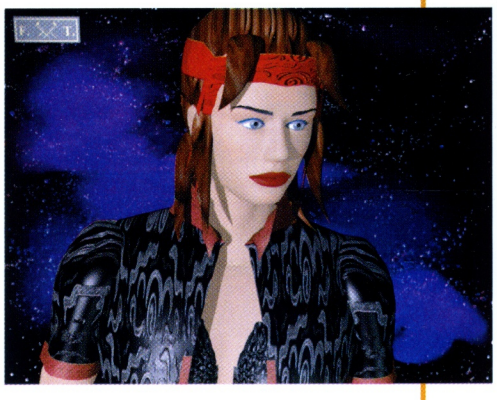
1) Tus problemas con «Imagine» pueden deberse a tu configuración de memoria. Tu versión de «Imagine» no funcionará desde una caja de Windows.

2) El tema de los VRML es muy interesante pero no se si realmente encajaría aquí. Llevo ya algún tiempo pensándolo.

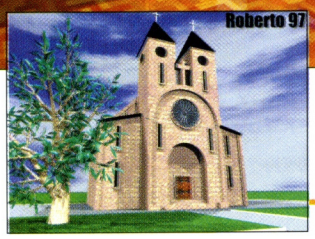
3) Lo del museo virtual con todos los modelos de los lectores me parece sencillamente impracticable. ¿Te das cuenta del trabajo que representaría? ¿De la cantidad de memoria y tiempo de cálculo que se precisarían? No me gusta decir "no" pero esta vez lo haré: no, no, no y no.



Dos nuevos autores que sólo se identifican como **F y T** nos han enviado unas impactantes escenas con personajes creados con una versión recortada de «3D Studio 3.0». Los tres son muy realistas y suponen un ejemplo perfecto de lo que puede conseguirse con la edición de vértices y... la paciencia. No veo nada que criticar en ellos como no sea su falta de expresividad. Tan sólo puedo aconsejaros que intentéis hacer lo mismo con un buen modelador de splines. A petición vuestra evitaremos la publicación de las mallas de los personajes.



El Foro del Lector



Roberto de los Ojos Gracia es otro povmaníaco que ya ha pasado anteriormente por estas páginas. Hoy Roberto nos envía diversas escenas entre las que destaca la habitación (por sus estupendas texturas) y la iglesia (aunque el tamaño del árbol que has puesto queda desproporcionado con respecto a aquella). En cuanto a tus problemas con #while, ¿has probado a emplear bucles anidados o a meter varias ventanas dentro de los bucles? En cuanto a tus preguntas y peticiones...

1) POV 3.0 no puede reproducir exactamente los mismo efectos que tiene el POVAFX. Habrá que esperar a que nuestro amigo Marcos tenga tiempo de implementar sus efectos en la actual versión de POV, pero eso, dado que actualmente está más liado que la pata de un romano, no parece muy probable por ahora.

2) Incluiremos nuevas bitmaps de tanto en tanto, cuando el tema tratado así lo exija.

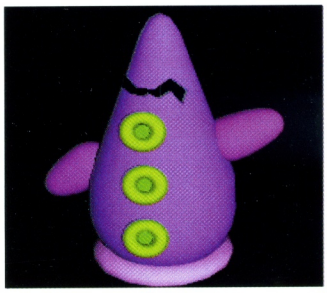
3) ¿Nuevos ficheros Dem? Bueno, quizá, no se...

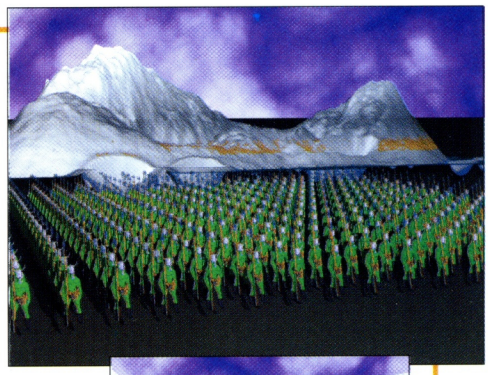
Oriol Bagan Carrasco nos envía varios proyectos creados con «Imagine 4.0» de los que pide crítica. A este respecto creo que como primer paso deberás practicar mucho con el modelador para que tus escenas ganen en complejidad y realismo. En cuanto a tus dudas te diré que...

1) Está pendiente un especial sobre algunos apartados de «Imagine».

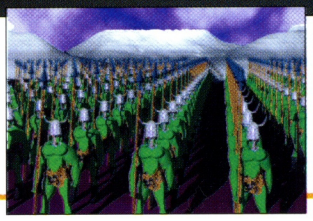
2) El diseño de las nubes dependerá del programa utilizado (otra cuestión a la que dedicar un artículo). Muchos autores suelen superponer objetos a los que se da un valor de transparencia y un color adecuado. También pueden emplearse bitmaps de fondo o utilizar algún efecto especial del motor si el programa dispone de él.

3) ¡POV no es difícil!





Antonio Antón, un profesor de matemáticas atacado también por el virus infográfico decidió comprobar por sí mismo la potencia de la sentencia mesh comentada en el número 14 de Rendermanía y para ello preparó su propio ejército de orcos. ¡Bien hecho, pero no te pares aquí! ¿Qué te parecen las posibilidades del lenguaje escénico de POV para crear variaciones en cada personaje del ejército? Espero verte de nuevo por aquí.



Bartolomé Colom Bauza nos envía una preciosa escena en la que un Dragón y una chica creados con «3D Studio» se recortan sobre un cielo verde. Bartolomé dice estar interesado en utilidades reductoras de polígonos (¿y quién no?). ¿Conoces el optimize.pxp de Yost Group? Este Ipas es la única opción que conozco para «3D Studio», aunque no me convence demasiado su algoritmo de funcionamiento. Gracias por tus mallas.